UNIT 4 EXPRESSIONS AND OPERATORS

Structure

4.0 INTRODUCTION

In the previous unit we have learnt variables, constants, datatypes and how to declare them in C programming. The next step is to use those variables in expressions. For writing an expression we need operators along with variables. An *expression* is a sequence of operators and operands that does one or a combination of the following:

- specifies the computation of a value
- designates an object or function
- generates side effects.

An *operator* performs an operation (evaluation) on one or more operands. An *operand* is a subexpression on which an operator acts.

This unit focuses on different types of operators available in C including the syntax and use of each operator and how they are used in C.

A computer is different from calculator in a sense that it can solve logical expressions also. Therefore, apart from arithmetic operators, C also contains logical operators. Hence, logical expressions are also discussed in this unit.

4.1 OBJECTIVES

After going through this unit you will be able to:

- write and evaluate arithmetic expressions;
- express and evaluate relational expressions;
- write and evaluate logical expressions;
- write and solve compute complex expressions (containing arithmetic, relational and logical operators), and
- check simple conditions using conditional operators.

4.2 ASSIGNMENT STATEMENT

In the previous unit, we have seen that variables are basically memory locations and they can hold certain values. But, how to assign values to the variables? C provides an assignment operator for this purpose. The function of this operator is to assign the values or values in variables on right hand side of an expression to variables on the left hand side.

The syntax of the assignment expression is as follows:

Variable = constant / variable/ expression;

The data type of the variable on left hand side should match the data type of constant/variable/expression on right hand side with a few exceptions where automatic type conversions are possible. Some examples of assignment statements are as follows:

```
b  = a ;       /* b is assigned the value of a */
b = 5 ;       /* b is assigned the value 5*/
b = a+5;    /* b is assigned the value of expr  a+5 */
```

The expression on the right hand side of the assignment statement can be:

- an arithmetic expression;
- a relational expression;
- a logical expression;
- a mixed expression.

The above mentioned expressions are different in terms of the type of operators connecting the variables and constants on the right hand side of the variable. Arithmetic operators, relational operators and logical operators are discussed in the following sections.

For example,

```
        int a;
        float b,c ,avg, t;
        avg = (b+c) / 2;              /*arithmetic expression */
        a = b && c;                    /*logical expression*/
        a = (b+c) && (b<c);          /* mixed  expression*/
```

4.3 ARITHMETIC OPERATORS

The basic arithmetic operators in C are the same as in most other computer languages, and correspond to our usual mathematical/algebraic symbolism. The following arithmetic operators are present in C:

Operator	**Meaning**
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modular Division

Some of the examples of algebraic expressions and their C notation are given below:

Expression	**C notation**
$\frac{b*g}{d}$	(b *g) / d
a^3+cd	(a*a*a) + (c*d)

The arithmetic operators are all binary operators i.e. all the operators have two operands. The integer division yields the integer result. For example, the expression 10/3 evaluates to 3 and the expression 15/4 evaluates to 3. C provides the modulus operator, %, which yields the reminder after integer division. The modulus operator is an integer operator that can be used only with integer operands. The expression x%y yields the reminder after x is divided by y. Therefore, 10%3 yields 1 and 15%4 yields 3. An attempt to divide by zero is undefined on computer system and generally results in a run- time error. Normally, Arithmetic expressions in C are written in straight-line form. Thus 'a divided by b' is written as a/b.

The operands in arithmetic expressions can be of integer, float, double type. In order to effectively develop C programs, it will be necessary for you to understand the rules that are used for implicit conversation of floating point and integer values in C.

They are mentioned below:

- An arithmetic operator between an integer and integer always yields an integer result.
- Operator between float and float yields a float result.
- Operator between integer and float yields a float result.

If the data type is double instead of float, then we get a result of double data type.

For example,

Operation	**Result**
5/3	1
5.0/3	1.3
5/3.0	1.3
5.0/3.0	1.3

Parentheses can be used in C expression in the same manner as algebraic expression For example,

```
a * (b + c).
```

It may so happen that the type of the expression and the type of the variable on the left hand side of the assignment operator may not be same. In such a case the value for the expression is promoted or demoted depending on the type of the variable on left hand side of = (assignment operator). For example, consider the following assignment statements:

```
int   i;
float b;
i = 4.6;
b = 20;
```

In the first assignment statement, float (4.6) is demoted to int. Hence *i* gets the value 4. In the second statement int (20) is promoted to float, *b* gets 20.0. If we have a complex expression like:

```
float   a, b, c;
int   s;
s = a * b / 5.0 * c;
```

Where some operands are integers and some are float, then int will be promoted or demoted depending on left hand side operator. In this case, demotion will take place since s is an integer.

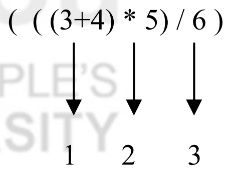
The rules of arithmetic precedence are as follows:

1. Parentheses are at the "highest level of precedence". In case of nested parenthesis, the innermost parentheses are evaluated first.

For example,

(((3+4)*5)/6)

The order of evaluation is given below.

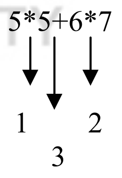


2. Multiplication, Division and Modulus operators are evaluated next. If an expression contains several multiplication, division and modulus operators, evaluation proceeds from left to right. These three are at the same level of precedence.

For example,

5*5+6*7

The order of evaluation is given below.

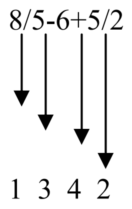


3. Addition, subtraction are evaluated last. If an expression contains several addition and subtraction operators, evaluation proceeds from left to right. Or the associativity is from left to right.

For example,

8/5-6+5/2

The order of evaluation is given below.



Apart from these binary arithmetic operators, C also contains two unary operators referred to as increment (++) and decrement (--) operators, which we are going to be discussed below:
The two-unary arithmetic operators provided by C are:

- ***Increment operator* (++)**
- ***Decrement operator* (- -)**

The increment operator increments the variable by one and decrement operator decrements the variable by one. These operators can be written in two forms i.e. before a variable or after a variable. If an ***increment / decrement*** operator is written before a variable, it is referred to as ***preincrement / predecrement*** operators and if it is written after a variable, it is referred to as ***post increment / postdecrement*** operator.

For example,

a++ or ++a is equivalent to a = a+1 and
a-- or - -a is equivalent to a = a -1

The importance of ***pre*** and ***post*** operator occurs while they are used in the expressions. ***Preincrementing (Predecrementing)*** a variable causes the variable to be incremented (decremented) by 1, then the new value of the variable is used in the expression in which it appears***. Postincrementing (postdecrementing)*** the variable causes the current value of the variable is used in the expression in which it appears, then the variable value is incremented (decrement) by 1.

The explanation is given in the table below:

Expression	**Explanation**
++a	Increment a by 1, then use the new value of a
a++	Use value of a, then increment a by 1
--b	Decrement b by 1, then use the new value of b
b--	Use the current value of b, then decrement by 1

The precedence of these operators is right to left. Let us consider the following examples:

```
int a = 2, b=3;
int c;
c = ++a – b- -;
printf (“a=%d, b=%d,c=%d\n”,a,b,c);
```

OUTPUT

```
a = 3, b = 2, c = 0.
```

Since the precedence of the operators is right to left, first b is evaluated, since it is a post decrement operator, current value of b will be used in the expression i.e. 3 and then b will be decremented by 1.Then, a preincrement operator is used with a, so first a is incremented to 3. Therefore, the value of the expression is evaluated to 0.

Let us take another example,

```
int a = 1, b = 2, c = 3;
int k;
```

```
k = (a++)*(++b) + ++a - --c;
printf("a=%d,b=%d, c=%d, k=%d",a,b,c,k);
```

OUTPUT

```
 a = 3, b = 3, c = 2, k = 6
```

The evaluation is explained below:

```
k  =  (a++) * (++b)+ ++a  - --c
   =  (a++) * (3) + 2 - 2     step1
   =  (2) * (3) + 2 - 2       step2
   =  6                       final result
```

Check Your Progress 1

1. Give the C expressions for the following algebraic expressions:

   i) $\frac{a*4c^2 - d}{m+n}$

   ii) $ab - (e+f)\frac{4}{c}$

   ……………………………………………………………………………………………

   ……………………………………………………………………………………………

2. Give the output of the following C code:

```
main()
{
   int a=2,b=3,c=4;
   k = ++b +  --a*c + a;
   printf("a= %d b=%d c=%d k=%d\n",a,b,c,k);
}
```

   ……………………………………………………………………………………………

   ……………………………………………………………………………………………

3. Point out the error:

```
Exp = a**b;
```

   ……………………………………………………………………………………………

   ……………………………………………………………………………………………

4.4 RELATIONAL OPERATORS

Executable C statements either perform actions (such as calculations or input or output of data) or make decision. Using relational operators we can compare two variables in the program. The C relational operators are summarized below, with their meanings. Pay particular attention to the equality operator; it consists of two equal signs, not just one. This section introduces a simple version of C's **if** control structure that allows a program to make a decision based on the result of some condition. If the condition is true then the statement in the body of if statement is executed else if the condition is false, the statement is not executed. Whether the body statement is executed or not, after the if structure completes, execution proceeds with the next statement after the if structure. Conditions in the **if** structure are formed with the relational operators which are summarized in the Table 4.1.

Table 1: Relational Operators in C

Relational Operator	Condition	Meaning
==	x==y	x is equal to y
!=	x!=y	x is not equal to y
<	x<y	x is less than y
<=	x<=y	x is less than or equal to y
>	x>y	x is greater than y
>=	x>=y	x is greater or equal to y

Relational operators usually appear in statements which are inquiring about the truth of some particular relationship between variables. Normally, the relational operators in C are the operators in the expressions that appear between the parentheses.
For example,

(i) if (thisNum < minimumSoFar) minimumSoFar = thisNum

(ii) if (job == Teacher) salary == minimumWage

(iii) if (numberOfLegs != 8) thisBug = insect

(iv) if (degreeOfPolynomial < 2) polynomial = linear

Let us see a simple C program containing the If statement (will be introduced in detail in the next unit). It displays the relationship between two numbers read from the keyboard.

Example: 4.1

```
/*Program to find relationship between two numbers*/

#include <stdio.h>
main ( )
{

int a, b;
printf ( “Please enter two integers: ”);
scanf (“%d%d”, &a, &b);
if (a <= b)
printf (“ %d <= %d\n”,a,b);
else
printf (“%d > %d\n”,a,b);
}
```

OUTPUT

```
Please enter two integers: 12 17
12 <= 17
```

We can change the values assigned to a and b and check the result.

4.5 LOGICAL OPERATORS

Logical operators in C, as with other computer languages, are used to evaluate expressions which may be true or false. Expressions which involve logical operations are evaluated and found to be one of two values: **true or false**. So far we have studied simple conditions. If we want to test multiple conditions in the process of making a decision, we have to perform simple tests in separate IF statements(will be introduced in detail in the next unit). C provides logical operators that may be used to form more complex conditions by combining simple conditions.

The logical operators are listed below:

Operator	**Meaning**
&&	Logical AND
||	Logical OR
!	Logical NOT

Thus logical operators (AND and OR) combine two conditions and logical NOT is used to negate the condition i.e. if the condition is true, NOT negates it to false and vice versa.Let us consider the following examples:

(i) Suppose the grade of the student is 'B' only if his marks lie within the range 65 to 75,if the condition would be:

```
if ((marks >=65) && (marks <= 75))
printf ("Grade is B\n");
```

(ii) Suppose we want to check that a student is eligible for admission if his PCM is greater than 85% or his aggregate is greater than 90%, then,

```
if ((PCM >=85) ||(aggregate >=90))
printf ("Eligible for admission\n");
```

Logical negation (!) enables the programmer to reverse the meaning of the condition. Unlike the && and || operators, which combines two conditions (and are therefore Binary operators), the logical negation operator is a unary operator and has one single condition as an operand. Let us consider an example:

```
if !(grade=='A')
printf ("the next grade is %c\n", grade);
```

The parentheses around the condition grade==A are needed because the logical operator has higher precedence than equality operator. In a condition if all the operators are present then the order of evaluation and associativity is provided in the table. The truth table of the logical AND (&&), OR (||) and NOT (!) are given below.

These table show the possible combinations of zero (false) and nonzero (true) values of x (expression1) and y (expression2) and only one expression in case of NOT operator. The following table 4.2 is the truth table for && operator.

Table 4. 2: Truth table for && operator

x	**y**	**x&&y**
zero	zero	0
Non zero	zero	0
zero	Non zero	0
Non zero	Non zero	1

The following table 4.3 is the truth table for || operator.

Table 4.3: Truth table for || operator

x	y	x || y
zero	zero	0
Non zero	zero	1
zero	Non zero	1
Non zero	Non zero	1

The following table 4.4 is the truth table for ! operator.

Table 4.4: Truth table for ! operator

x	! x
zero	1
Non zero	0

The following table 4.5 shows the operator precedence and associativity

Table 4.5: (Logical operators precedence and associativity)

Operator	Associativity
!	Right to left
&&	Left to right
||	Left to right

4.6 COMMA AND CONDITIONAL OPERATORS

Conditional Operator

C provides an called as the conditional operator (**?:**) which is closely related to the **if/else** structure. The conditional operator is C's only ternary operator - it takes three operands. The operands together with the conditional operator form a conditional expression. The first operand is a condition, the second operand represents the value of the entire conditional expression it is the condition is true and the third operand is the value for the entire conditional expression if the condition is false.

The syntax is as follows:

(condition)? (expression1): (expression2);

If condition is true, expression1 is evaluated else expression2 is evaluated. Expression1/Expression2 can also be further conditional expression i.e. the case of nested if statement (will be discussed in the next unit).

Let us see the following examples:

(i) x= (y<20) ? 9: 10;
This means, if (y<20), then x=9 else x=10;

(ii) printf ("%s\n", grade>=50? "Passed": "failed");
The above statement will print "passed" grade>=50 else it will print "failed"

(iii) (a>b) ? printf ("a is greater than b \n"): printf ("b is greater than a \n");

If a is greater than b, then first printf statement is executed else second printf statement is executed.

Comma Operator

A comma operator is used to separate a pair of expressions. A pair of expressions separated by a comma is evaluated left to right, and the type and value of the result are the value of the type and value of the right operand. All side effects from the evaluation of the left operand are completed before beginning evaluation of the right operand. The left side of comma operator is always evaluated to void. This means that the expression on the right hand side becomes the value of the total comma-separated expression. For example,

```
x = (y=2, y - 1);
```

first assigns y the value 2 and then x the value 1. Parenthesis is necessary since comma operator has lower precedence than assignment operator.

Generally, comma operator (,) is used in the for loop (will be introduced in the next unit)

For example,

```
for (i = 0,j = n;i<j; i++,j--)
{
  printf ("A");
}
```

In this example **for** is the looping construct (discussed in the next unit). In this loop, i = 0 and j = n are separated by comma (,) and i++ and j—are separated by comma (,). The example will be clear to you once you have learnt for loop (will be introduced in the next unit).

Essentially, the comma causes a sequence of operations to be performed. When it is used on the right hand side of the assignment statement, the value assigned is the value of the last expression in the comma-separated list.

Check Your Progress 2

1. Given a=3, b=4, c=2, what is the result of following logical expressions:
   (a < --b) && (a==c)

   ……………………………………………………………………………………

   ……………………………………………………………………………………

2. Give the output of the following code:

```
main()
{
  int a=10, b=15,x;
```

```
    x = (a<b)?++a:++b;
    printf("x=%d a=%d b=%d\n",x,a,b);
  }
```

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

3. What is the use of comma operator?

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

4.7 TYPE CAST OPERATOR

We have seen in the previous sections and last unit that when constants and variables of different types are mixed in an expression, they are converted to the same type. That is automatic type conversion takes place. The following type conversion rules are followed:

1. All chars and **short ints** are converted to **ints**. All floats are converted to doubles.

2. In case of binary operators, if one of the two operands is a **long double**, the other operand is converted to **long double**,

   else if one operand is **double**, the other is converted to **double**,
   else if one operand is **long**, the other is converted to **long**,
   else if one operand is **unsigned**, the other is converted to **unsigned**,

C converts all operands "up" to the type of largest operand (largest in terms of memory requirement for e.g. **float** requires 4 bytes of storage and **int** requires 2 bytes of storage so if one operand is **int** and the other is **float**, **int** is converted to **float**).

All the above mentioned conversions are automatic conversions, but what if **int** is to be converted to **float.** It is possible to force an expression to be of specific type by using operator called a ***cast***. The syntax is as follows:

(type) expression

where *type* is the standard C data type. For example, if you want to make sure that the expression a/5 would evaluate to type **float** you would write it as

(float) a/5

cast is an unary operator and has the same precedence as any other unary operator. The use of ***cast*** operator is explained in the following example:

```
 main()
 {
    int   num;
    printf("%f %f %f\n", (float)num/2, (float)num/3, float)num/3);
  }
```

Tha ***cast*** operator in this example will ensure that fractional part is also displayed on the screen.

4.8 SIZE OF OPERATOR

C provides a compile-time unary operator called ***sizeof*** that can be used to compute the size of any object. The expressions such as:

sizeof object and ***sizeof(type name)***

result in an unsigned integer value equal to the size of the specified object or type in bytes. Actually the resultant integer is the number of bytes required to store an object of the type of its operand. An object can be a variable or array or structure. An array and structure are data structures provided in C, introduced in latter units. A type name can be the name of any basic type like **int** or **double** or a derived type like a structure or a pointer.

For example,

```
    sizeof(char)  =  1bytes
    sizeof(int) = 2 bytes
```

4.9 C SHORTHAND

C has a special shorthand that simplifies coding of certain type of assignment statements. For example:

```
a = a+2;
```

can be written as:

```
a += 2;
```

The operator +=tells the compiler that a is assigned the value of a + 2;
This shorthand works for all binary operators in C. The general form is:

variable operator = variable / constant / expression

These operators are listed below:

Operators	Examples	Meaning
+=	a+=2	a=a+2
-=	a-=2	a=a-2
=	**a***=2	a = a*2
/=	a/=2	a=a/2
%=	a%=2	a=a%2

Operators	Examples	Meaning
&&=	a&&=c	a=a&&c
||=	a||=c	a=a||c

4.10 PRIORITY OF OPERATORS

Since all the operators we have studied in this unit can be used together in an expression, C uses a certain hierarchy to solve such kind of mixed expressions. The hierarchy and associatively of the operators discussed so far is summarized in Table 6. The operators written in the same line have the same priority. The higher precedence operators are written first

Table 4.6: Precedence of the operators

Operators	Associativity
( )	Left to right
! ++ -- (*type*) sizeof	Right to left
/ %	Left to right
+ -	Left to right
< <= > >=	Left to right
== !=	Left to right
&&	Left to right
||	Left to right
?:	Right to left
= += -= *= /= %= &&= ||=	Right to left
,	Left to right

Check Your Progress 3

1. Give the output of the following C code:

```
main( )
{
    int a,b=5;
    float f;

     a=5/2;
     f=(float)b/2.0;
     (a<f)? b=1:b=0;
     printf("b = %d\n",b);
}
```

……………………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

2. What is the difference between && and &. Explain with an example.

……………………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

3. Use of Bit Wise operators makes the execution of the program.

……………………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

4.11 SUMMARY

In this unit, we discussed about the different types of operators, namely arithmetic, relational, logical present in C and their use. In the following units, you will study how these are used in C's other constructs like control statements, arrays etc.

This unit also focused on type conversions. Type conversions are very important to understand because sometimes a programmer gets unexpected results (logical error) which are most often caused by type conversions in case user has used improper types or if he has not type cast to desired type.

This unit also referred to C shorthand. C is referred to as a compact language which is because lengthy expressions can be written in short form. Conditional operator is one of the examples, which is the short form of writing the if/else construct (next unit). Also increment/decrement operators reduce a bit of coding when used in expressions.

Since Logical operators are used further in all types of looping constructs and if/else construct (in the next unit), they should be thoroughly understood.

4.12 SOLUTIONS / ANSWERS

Check Your Progress 1

1. C expression would be

   i) ((a*4*c*c)-d)/(m+n)
   ii) a*b-(e+f)*4/c

2. The output would be:
   a=1 b=4 c=4 k=10

3. There is no such operator as **.

Check Your Progress 2

1. The expression is evaluated as under:

```
(3 < - -4) && (3== 2)
 (3 < 3) &&  (3==2)
 0  && 0
 0
```

   Logical false evaluates to 0 and logical true evaluates to 1.

2. The output would be as follows:

   x=11, a=11, b=16

3. Comma operator causes a sequence of operators to be performed.

Check Your Progress 3

1. Here a will evaluate to 2 and f will evaluate to 2.5 since type cast operator is used in the latter so data type of b changes to float in an expression. Therefore, output would be b=1.

2. && operator is a logical and operator and & is a bit wise and operator. Therefore, && operator always evaluates to true or false i.e 1 or 0 respectively while & operator evaluates bit wise so the result can be any value. For example:

```
2 && 5 => 1(true)
2 &  5 => 0(bit-wise anding)
```

3. Use of Bit Wise operators makes the execution of the program faster.

4.13 FURTHER READINGS

1. The C Programming Language, *Kernighan & Richie,* PHI Publication**.**
2. Computer Science A structured programming approach using C*, Behrouza A. Forouzan, Richard F. Gilberg, Second Edition, Brooks/Cole*, Thomson Learning, 2001.
3. Programming with C*,* Second Edition, *Byron Gottfried*, Schaum Outline, Tata Mc Graw Hill, 2003.